Programming in Prolog – List of Exercises #4

1. Consider a graph with costs associated with each arc. Write a Prolog program that can find a path of minimum cost between node A and node B. (You can invent your own graph or use some actual example: map of cities with their distances).

2. Define the predicate `height(BinaryTree,Height)` that can return the height of a binary tree. The height of an empty tree is zero and the height of a tree with one element is 1.

3. Define a predicate that can recognize if a Prolog term is a list.

4. Predicates `sub1`, `sub2` e `sub3`, below, implement a relation over lists. The `sub1` predicate has a more procedural definition than `sub2` and `sub3`. These last two are written in a more declarative way. Observe the behavior of these predicates with respect to efficiency. Two of them have similar efficiency. Which ones? Why one of them is inefficient?

```
sub1(List,Sublist) :- prefix(List,Sublist).
sub1([_|Tail],Sublist) :- sub1(Tail,Sublist).

prefix(_,[]).
prefix([X|List1],[X|List2]) :- prefix(List1,List2).

sub2(List,Sublist) :- conc(List1,List2,List),
                      conc(List3,Sublist,List1).

sub3(List,Sublist) :- conc(List1,List2,List),
                      conc(Sublist,_,List2).
```

5. Define the relation `reverse(List,RevList)`, where the arguments are represented as list differences.

6. Use the **bagof/3** predicate to define the relation `powerset(Set,Subsets)` that compute the set of all sets. (represent the sets as lists)

7. Define the relation `alv(Tree)` to test if a binary tree is AVL, i.e., all subtrees can not differ in depth more than 1 level. Represent the binary tree using the Prolog term: `t(Left,Root,Right)` or `nil` if the subtree is empty.

8. How the search programs given in class could be modified to perform the search starting from multiple initial states instead of just one?